Application Programming in Com-plete

This chapter covers the following topics:

- How Com-plete Works
- Specific Programming Language Considerations

How Com-plete Works

Online programs are initiated by request of a terminal user or by request of an executing online program. The terminal user must type

```
*pgmname  parameters
```

or invoke a program via the COM-PASS menu. In either case, **pgmname parameters*, including any terminal-dependent control characters, is placed into a terminal buffer associated with that terminal. Com-plete searches for the load module *pgmnam*, allocates a thread, places the program into the thread (if the program is not defined as RESIDENTPAGE or does not reside in the LPA/SVE), and then passes control to that program. This root program can use those Com-plete functions available to it. The root program terminates either by calling a Com-plete termination function (EOJ or WRTxD) or by returning via the language-specific return mechanism (e.g., stop run, go back, return, etc.).

Threads

During execution, online programs reside in areas called threads. User threads are fixed in size and range from a minimum of 8K to a maximum of 1008K below the 16 MB line. For each thread, a fixed-size extension can be allocated above the 16 MB line. Note that load modules linked with OVERLAY cannot be used with Com-plete.

Rolling Mechanism

Com-plete uses its own thread paging supervisor. Whenever an application program (or Com-plete utility) running in a user thread issues a request for a terminal operation, ADABAS or ROLOUT function, the Com-plete paging supervisor can be invoked. If another user is waiting for the thread, the entire contents of the thread are paged out (ROLOUT) to a paging data set and the waiting application is paged in (ROLIN). Because of the mechanics of the thread paging supervisor, each terminal can use its own copy of a program and alleviate the need for writing reentrant programs.

Note that during a ROLOUT operation, the program/thread counters CPU time, REAL time and ADABAS calls are reset.

COM-PASS Considerations

All programs that reside in the Com-plete program library can be invoked by the terminal user. The following points should be considered to ensure that programs are not used by unauthorized users:

- Main programs should be protected by the Com-plete Security System;

- A subprogram that is COLOADed will not be security protected;
- Any routine that is invoked from a terminal will always contain an ’*’ at the start of the input data buffer;
- COM-PASS checks the integrity of transactions only if the data buffer that is transferred with the transaction begins with an asterisk (*);
- A main program that fetches or attaches another should transfer a data buffer that does not start with an asterisk (*) in order to distinguish it from a terminal call.

COM-PASS allows up to nine transactions per user to be defined under the control of transaction security. This assumes that the User Profile definition option ’ALLOWED NON-MENU PROGRAMS’ is specified as ’NO’.

Write Conversational Calls

Each transaction within COM-PASS can be suspended whenever the transaction contains a write conversational (WRTC) call to Com-plete. COM-PASS provides a return code of 16 when the transaction is restarted. To ensure the correct use of the COM-PASS Restart/Suspend facilities, each transaction that contains a write conversational call should check for an RC=16. Programs that use writes without erase should rebuild the entire screen whenever the program has been suspended.

Specific Programming Language Considerations

Each argument in a call to Com-plete must be defined with the data type required for that argument. The data types used are:

Alphanumeric	Fixed-length fields representing character data.NATURAL - (An). COBOL - PICTURE X(n). PL/I - CHARACTER (n). Assembler - DC CLn’ ’.
Binary fullword	Four-byte fields representing a signed binary value. NATURAL - (B4). COBOL - PICTURE S9(8) COMP. PL/I - FIXED BINARY (31). Assembler - DC F’n’.
Binary halfword	Two-byte fields representing a signed binary value. NATURAL - (B2). COBOL - PICTURE S9(4) COMP. PL/I - FIXED BINARY(15). Assembler - DC H’n’.

NATURAL

In general, NATURAL provides Com-plete functions to the programmer that are built into the NATURAL language. For example, terminal I/O functions of Com-plete are implemented via the NATURAL INPUT verb. Applications written in NATURAL should use the built-in facilities of NATURAL wherever possible.

For access to a Com-plete function not directly available in the NATURAL language, code a CALL statement to the Com-plete subroutine with the function’s name (for example, SDOPEN for SD file open). The NATURAL CALL statement is equivalent to the function of COLINK; therefore, COLINK should never be called from NATURAL.

Com-plete functions are available to the NATURAL CALL statement, if those subroutines are cataloged in the Com-plete program library or are placed into the RESIDENTPAGE portion of Com-plete.

COBOL

MVS COBOL programs should be compiled with options ’NOSTAE, NOSTATE, and ENDJOB’.

VSE COBOL programs should be compiled with control card:

```
CBL NOSTXIT,NOSTATE,NOCOUNT
```

and must not use LFOW or SYMDMP.

COBOL II

COBOL II enables you to generate reentrant code for the COBOL programs. This facility can be used to its optimum effect with Com-plete. Programs that are reentrant can run as RESIDENTPAGE programs within the Com-plete nucleus. Com-plete can also handle programs that run above the 16 MB line. Also, the COBOL run time routines can be linked into one COBPACK which can also be RESIDENTPAGE or in LPA. Com-plete can support this both below and above the 16 MB line as applicable.

You must choose the IGZTUNE parameters that these programs use when running under Com-plete. In particular the INIBLOW and the INIABOV parameters, which indicate the amount of storage COBOL’s storage management routines will receive at startup before doing anything else. As this storage is allocated from thread, at least this amount must be available in the thead before the COBOL program can run. If this storage is never used, then valuable thread and rollbuffer space will be wasted. We therefore recommend that these are set to reflect the storage usage of the most of your COBOL programs (90% or more) and let COBOL request the storage for the rest of the programs. This will simply result in a few more getmain/freemain requests from the COBOL programs for which the allocated space is not enough.

Care should also be taken with the INIABOV parameter, as this storage is also allocated from thread and therefore is not allocated above the 16 MB line. COBOL expects this GETMAIN to always work and so, having issued a conditional GETMAIN to get the storage, proceeds to try to use the storage which may not be there if the GETMAIN fails (as can happen under Com-plete if the catalog size for the program is not large enough). This eventually results in a storage exception abend when COBOL tries to address storage which is not addressable by your program. For this reason, we recommend that if unexplainable COBOL addressing exceptions occur, this parameter should be set to 1, the Com-plete catalog size for the program increased and the program tried once more.

PL/I

With the exception of the COLINK and COXCTL functions, Com-plete does not support PL/I dope vectors; therefore, the declaration of the entry for a Com-plete function must specify the ASM option. In addition, the EXTERNAL attribute must always be specified.

The ’NOREPORT’ compile option must not be used.

The COLINK and COXCTL functions must be defined with the attributes of the target program.

The following definition is recommended:

```
DCL function ENTRY EXTERNAL OPTIONS (ASM,INTER);
DCL IRETURN FIXED BIN(31) INITIAL (0);
DCL FIELD CHAR (10) INITIAL ('CONSTANT');
CALL function (IRETURN, FIELD);
```

When calling a PL/1 subroutine from a PL/1 main program, the subroutine needs to be linked with the entry point name the same as the subroutine name. This obtains the correct entry and prevents PL/1 repeatedly setting up the run-time environment. See the *PL/1 Optimising Compiler Programmer's Guide* for further information.

FORTRAN Notes

FORTRAN routines require privileged status. Catalog FORTRAN programs with the ULIB PV operand (see the Com-plete Utilities documentation).

Assembler

CM$CALL

Assembler programmers have a choice of using the CALL macro distributed and documented for their operating system or a CM$CALL macro distributed with Com-plete. The CM$CALL macro manages operating system independent calls.

The format of the CM$CALL is:

```
CM$CALL ENTRY,PARMS,PLIST=
```

where:

ENTRY	Required.
	Name of routine. If (R15) is used, register 15 must have the address of the subroutine.
PARMS	Optional.
	A list of parameters in parentheses separated by commas or null. If a list is specified, each parameter must be a valid second operand of a load address (LA) instruction.
PLIST	Optional.
	A valid second operand of a load address (LA) instruction or a parenthesized register specification.

The parameter list for the CALL will be:

1. Null, if PARMS and PLIST are both absent.

2. Generated inline, if PARMS is specified, but not PLIST.

3. Generated at a location specified by PLIST, if PARMS is specified.

4. Assumed to exist at a location specified by PLIST, if PARMS are not specified. The end-of-parameter-list indicator must be set.

Be aware that Com-plete validates the parameter list. The last item in the parameter list must have the end-of-parameter-list indicator set. The LV option must be specified in the MVS CALL macro in order to set this indicator.